Fast Bayesian Optimization of Function Networks with Partial Evaluations

Anonymous[1]

[1]Anonymous Institution

Abstract Bayesian optimization of function networks (BOFN) is a framework for optimizing expensive-to-evaluate objective functions structured as networks, where some nodes' outputs serve as inputs for others. Many real-world applications, such as manufacturing and drug discovery, involve function networks with additional properties—nodes that can be evaluated independently and incur varying costs. A recent BOFN variant, p-KGFN, leverages this structure and enables cost-aware partial evaluations, selectively querying only a subset of nodes at each iteration. However, despite its effectiveness, p-KGFN suffers from computational inefficiency due to its formulation that requires solving nested optimizations with a Monte Carlo-based objective function and the increasing number of acquisition function optimizations as the network size grows. To address this, we propose an accelerated p-KGFN algorithm that reduces computational overhead by considering a single acquisition function problem per iteration. This approach first generates candidate inputs for the entire network and leverages simulated intermediate outputs to form node-specific candidates. Experiments on benchmark problems show that our method maintains competitive performance while achieving up to a 16× speedup over existing BOFN methods.

1 Introduction

Bayesian Optimization (BO) (Jones et al., 1998; Frazier, 2018) stands out as a robust and efficient method for solving optimization problems of the form $x^* \in \arg\max_{x \in \mathcal{X}} f(x)$, where the objective function $f(x)$ is a time-consuming-to-evaluate derivative-free black-box function.

Starting with an initial set of n observations $D_n = \{(x_i, f(x_i)\}_{i=1}^n$, BO builds a surrogate model approximating the objective function $f(x)$. It then employs an acquisition function $\alpha_n(x)$, derived from the surrogate model, that quantifies the value of evaluating $f(x)$ at a new input point x. BO optimizes this acquisition function to choose the next input point x at which to evaluate the objective function. Once this point is evaluated, the newly obtained data is incorporated into the observation set and the process iterates until an evaluation budget is exhausted.

The BO framework has demonstrated remarkable success across a broad spectrum of real-world applications, including hyperparameter optimization in machine learning (Snoek et al., 2012), materials design (Frazier and Wang, 2016), model calibration (Sha et al., 2020), agricultural planning (Cosenza et al., 2022), and manufacturing processes (Deneault et al., 2021).

While treating the objective function as a black box makes BO easy to apply, recently emerging grey-box BO methods (Astudillo and Frazier, 2021b) aim to accelerate optimization by exploiting side information produced during objective function evaluations and by modifying the objective function evaluation itself.

Bayesian optimization of function networks (BOFN) (Astudillo and Frazier, 2019, 2021a) is a leading grey-box BO approach. BOFN considers objective functions $f(x)$ that are compositions of two or more black-box functions, as in Figure 1. Such compositions are called *function networks* and are described with a directed acyclic graph (DAG) where each node in the graph is a function and each edge is an input or output to/from a function. Function networks appear in real-world applications such as epidemic model calibration (Garnett, 2002), robotic control (Plappert et al.,

2018) and solar cell production (Kusakawa et al., 2022). BOFN builds surrogates for these individual constituent functions by observing inputs and outputs (so-called intermediate outcomes) obtained during objective function evaluations. It then uses this extra information to guide the selection of points to evaluate, accelerating optimization.

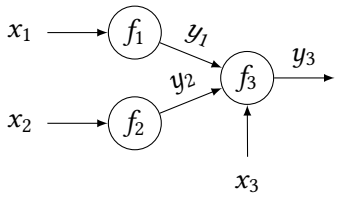

Figure 1: An example of an objective function modeled as a function network. The objective function's input is the vector $x \in \mathcal{X}$ comprised of three variables: x_1, x_2 and x_3. The objective is evaluated by evaluating individual functions f_1, f_2 and f_3 (shown as nodes in a directed acyclic graph) on their inputs (shown as edges in the graph). Its value is $f(x) = y_3(x) = f_3(f_1(x_1), f_2(x_2), x_3)$.

In the BOFN framework, Buathong et al. (2024) recently showed that optimization can be further accelerated by intelligently performing *partial evaluations*, i.e. evaluating only some of the functions in the function network in each iteration. In many applications, a partial evaluation is less time consuming than a full objective function evaluation and yet can provide high-value information. Using Figure 1 as an example, to optimize the final output y_3, one might evaluate f_1 at an input x_1 and observe $y_1 = f_1(x_1)$. This observation might suggest that x_1 is promising (if, for example, y_1 is large and our posterior suggests f_3 is increasing in y_1), in which case one might decide to evaluate f_3 at y_1 and some other previously-observed promising value for y_2. Or, if the observation suggests that x_1 is not promising, we might evaluate f_1 at a different input. Buathong et al. (2024) proposes an acquisition function, called the knowledge-gradient method for function networks with partial evaluations (p-KGFN), that guides the choice of individual functions to evaluate and the inputs at which to evaluate them by considering the value of the information obtained per unit evaluation cost.

While the p-KGFN acquisition function significantly reduces the time spent on objective function evaluation compared to previous BOFN and classical BO approaches, the acquisition function itself is time-consuming to evaluate and optimize. This makes the approach only useful in problems where the cost of objective function evaluation is so high that it outweighs the computational costs of acquisition function optimization, limiting its applicability. A key computational bottleneck is that computing p-KGFN requires solving a nested optimization problem with a Monte Carlo-based objective function over a mixed discrete/continuous search space: the set of nodes in the function network and their continuous vector-valued inputs.

To overcome this challenge and accelerate optimization across a much broader range of problems, we develop a fast-to-compute acquisition function, called Fast p-KGFN, that provides much of the benefit of p-KGFN at a fraction of the computational cost. Unlike the original p-KGFN, our approach avoids the need to solve a nested optimization problem. Instead, it uses a novel approach to identify a promising set of candidate measurements — candidate nodes and inputs at which to evaluate them. It then uses a novel approach for quickly evaluating the p-KGFN acquisition function for each candidate.

After introducing our new Fast p-KGFN approach, we validate the proposed algorithm performance across several test problems, demonstrating that it achieves comparable optimization results while significantly accelerating the original p-KGFN.

2 Problem Setting

We now formally describe function networks. Our presentation follows Astudillo and Frazier (2021a) and Buathong et al. (2024). We consider a sequence of functions $f_1, f_2, \ldots, f_K$ corresponding to nodes $\mathcal{V} = \{1, 2, \ldots, K\}$ in a DAG, $G = (\mathcal{V}, \mathcal{E})$. If an edge (i, j) appears in the DAG ($(i, j) \in \mathcal{E}$), this indicates that function i produces output that is consumed as input by function j. For simplicity, we assume each f_i produces scalar output though our approach generalizes easily to vector outputs.

Let $\mathcal{J}(k) = \{j : (j, k) \in E\}$ denote the set of parent nodes of node k, where j is said to be a parent node of k if k consumes input produced as output from j. We assume that nodes are ordered such that $j < k$ for all $j \in \mathcal{J}(k)$. We suppose that there is an external input to the function network, indicated by x and taking values in $\mathcal{X} \subseteq \mathbb{R}^d$. The output from the function network, and thus the objective function value, is determined by x. Let $\mathcal{I}(k) \subseteq \{1, \ldots, d\}$ be the set of components of x that are taken as input by function f_k.

With these definitions, the output at node f_k when the input to the function network is x is given by a recursive formula:

$$y_k(x) = f_k(y_{\mathcal{J}(k)}(x), x_{\mathcal{I}(k)}), \quad \forall k = 1, \ldots, K, \tag{1}$$

where $y_{\mathcal{J}(k)}(x)$ denotes a vector of outputs from node k's parent nodes, i.e. $y_{\mathcal{J}(k)}(x) = [y_j(x)]_{j \in \mathcal{J}(k)}$, and $x_{\mathcal{I}(k)} = [x_i]_{i \in \mathcal{I}(k)}$ are the external inputs to node k. We group the two types of inputs to f_k, $y_{\mathcal{J}(k)}(x)$ and $x_{\mathcal{I}(k)}$, into a single vector z_k. Then, the set of possible inputs to node k is $\mathcal{Z}_k = \mathcal{Y}_{\mathcal{J}(k)} \times \mathcal{X}_{\mathcal{I}(k)}$ where $\mathcal{Y}_{\mathcal{J}(k)}$ represents the set of all possible values for the parent nodes' outputs and $\mathcal{X}_{\mathcal{I}(k)}$ denotes the set of possible values for $x_{\mathcal{I}(k)}$.

Our goal is to adaptively choose nodes k and associated inputs z_k to learn a near-optimal input x to the function network that maximizes the output at the final node $y_K(x)$. For each node k, we assume an associated positive evaluation cost function $c_k(\cdot)$, and the learning task should be accomplished while minimizing the cumulative evaluation cost.

Buathong et al. (2024) allowed a restriction on the nodes and inputs evaluated that arises in some applications. In this restriction, when evaluating a function node f_k at input that includes node output $y_{\mathcal{J}(k)}$, it is necessary to first provide parent node evaluations that produce this output. We do not consider this restriction here, though we believe that our approach can be extended to applications where this restriction holds. Instead, for each k, we assume that a set containing $\mathcal{Y}_{\mathcal{J}(k)}$ is known and that f_k can be evaluated at any input in this set. This set containing $\mathcal{Y}_{\mathcal{J}(k)}$ could simply be $\mathbb{R}^{|\mathcal{J}(k)|}$ or it could be some strict subset.

3 Existing Methods

We now present existing methods relevant to our method, focusing on Astudillo and Frazier (2021a) and Buathong et al. (2024). We first present the approach to inference proposed in Astudillo and Frazier (2021a) and used in Buathong et al. (2024), and which we also use. We then present two acquisition functions that we build on in our work.

Inference. To perform inference, Astudillo and Frazier (2021a) proposes to model each function f_k with an independent Gaussian process (GP) (Williams and Rasmussen, 2006). This is tractable because our data is acquired as a collection of input/output pairs for each node k, $\mathcal{D}_{n_k(n),k} = \{(z_{i,k}, y_{i,k})\}_{i=1}^{n_k(n)}$, where $n_k(n)$ is the number observations at node k after the total of n evaluations. For simplicity, we will suppress the explicit dependency on n and will use the notation n_k throughout the manuscript. With data acquired in this way, the resulting posterior distributions on $f_1, \ldots, f_k$ remain conditionally independent GPs. We let $\mu_{n,k}(\cdot)$ and $\Sigma_{n,k}(\cdot, \cdot)$ denote the posterior mean and kernel associated with the GP on f_k given $\mathcal{D}_{n_k,k}$ at the time when a total of n evaluations have been performed.

Algorithm 1 Proposed Algorithm

Input:
The network DAG; Observation set $\mathcal{D}_n$; $c_k(\cdot)$, the evaluation cost function for node k, $k = 1, \ldots, K$; B, the total evaluation budget; $\mu_{n,k}$ and $\sigma_{n,k}$, the mean and standard deviation of the GP for node $k, k = 1, \ldots, K$ (fitted using initial observations $\mathcal{D}_n$);
Output: the point with the largest posterior mean at the final function node

$b \leftarrow 0$
while $b < B$ **do**
  Generate an EIFN network candidate $\hat{x}_n$ by solving Eq. (2)
  Sample a realization function $\hat{f}_k$ from a GP at node k, $\forall k = 1, \ldots, K$
  Construct a network realization $\hat{f}$ by combining $\hat{f}_k$, $\forall k = 1, \ldots, K$ according to the DAG
  Compute intermediate output $\hat{y}_k(\hat{x}_n)$ by a recursive formula similar to Eq. (1).
  Construct a node-specific input $\hat{z}_{n,k} = (\hat{y}_{\mathcal{J}(k)}(\hat{x}), \hat{x}_{\mathcal{I}(k)})$, $\forall k = 1, \ldots, K$
  Construct a discrete set $\mathcal{A}$ using Thompson sampling and local point methods.
  Evaluate a p-KGFN value, $\alpha_{n,k}(\hat{z}_{n,k})$, in Eq. (4) with using the discrete set $\mathcal{A}$, $\forall k = 1, \ldots, K$
  Select $\hat{k} \leftarrow \arg\max_{k=1,\ldots,K} \alpha_{n,k}(\hat{z}_{n,k})$
  Obtain the resulting evaluation $f_{\hat{k}}(\hat{z}_{n,\hat{k}})$
  Update the GP model for node $\hat{k}$ with the additional observation $(\hat{z}_{n,\hat{k}}, f_{\hat{k}}(\hat{z}_{n,\hat{k}}))$
  Update budget $b \leftarrow b + c_{\hat{k}}(\hat{z}_{n,\hat{k}})$ and iteration $n \leftarrow n + 1$
end while
return $\arg\max_{x \in \mathcal{X}} \nu_n(x)$ the maximum value of the posterior mean at the final node output

Let $\mathcal{D}_n = \cup_{k=1}^{K} \mathcal{D}_{n_k,k}$ be the combined observation set. The conditionally independent GP posterior distributions over $f_1, \ldots, f_K$ given $\mathcal{D}_n$ further induce a posterior distribution over the final node output $y_K(\cdot)$. However, due to its compositional network structure, this induced posterior distribution of $y_K(\cdot)$ is not Gaussian.

The EIFN Acquisition Function. Using the statistical model above, Astudillo and Frazier (2021a), proposed the EIFN acquisition function to select a candidate $\hat{x}_n \in \mathcal{X}$ at which to evaluate the entire function network. For example, this $\hat{x}_n$ in the Figure 1 example is a 3-dimensional input tuple $\hat{x}_n = (\hat{x}_{n,1}, \hat{x}_{n,2}, \hat{x}_{n,3})$. We refer to such evaluations of the entire function network as *full* evaluations, in contrast with our focus in this paper on partial evaluations. We introduce EIFN because we will use it as a tool in our approach.

To define the EIFN acquisition function, define $y^*_{n,K} = \max_{i=1,\ldots,n_K} y_K(x_i)$ as the current best observed value at the final node given $\mathcal{D}_n$. The EIFN at a proposed point x is the expected improvement of $y_K(x)$ over the current $y^*_{n,K}$ under the current posterior. Specifically,

$$\text{EIFN}_n(x) = \mathbb{E}_n[(y_K(x) - y^*_{n,K})^+ | \mathcal{D}_n], \tag{2}$$

where $(a)^+ = \max\{0, a\}$. The candidate selected is $\hat{x}_n \in \arg\max_{x \in \mathcal{X}} \text{EIFN}_n(x)$.

The p-KGFN Acquisition Function. Buathong et al. (2024) introduced the cost-aware *knowledge gradient for function networks with partial evaluations* (p-KGFN) acquisition function. This acquisition function proposes an individual candidate node k and an associated input $\hat{z}_k \in \mathcal{Z}_k$.

To define p-KGFN, let $\nu_n(x) = \mathbb{E}_n[y_K(x)|\mathcal{D}_n]$ be the posterior mean of the final node's output evaluated at network input x and define the maximum value of this current posterior mean function

$$\nu^*_n = \max_{x \in \mathcal{X}} \nu_n(x), \tag{3}$$

as the current solution quality. At each BO iteration, p-KGFN loops through all function nodes and proposes a candidate $\hat{z}_{n,k}$ that solves: $\hat{z}_{n,k} \in \arg\max_{z_k \in \mathcal{Z}_k} \alpha_{n,k}(z_k)$, where

$$\alpha_{n,k}(z_k) = \frac{\mathbb{E}_{y_k(z_k)}[\nu^*_{n+1} | z_k, y_k(z_k)] - \nu^*_n}{c_k(z_k)}. \tag{4}$$

The p-KGFN method proposes a node-specific candidate $\hat{z}_{n,k}$ that maximizes the the expected improvement of the solution quality per unit evaluation cost. Then the p-KGFN compares these node-specific candidates and selects the node $\hat{k} \in \arg\max_{k=1,\ldots,K} \alpha_{n,k}(\hat{z}_{n,k})$ to evaluate at its corresponding input $\hat{z}_{n,\hat{k}}$.

The p-KGFN acquisition function is challenging to optimize because it is defined by a complex nested expectation. Specifically, it is the expectation with respect to an unknown observation, $y_k(z_k)$, over the value of a random maximization problem. This maximization problem is itself challenging to evaluate because its objective function $\nu_n(x) = \mathbb{E}_n[y_K(x)|\mathcal{D}_n]$ is an expectation (under the updated posterior given $y_k(z_k)$). These expectations do not have analytical expressions and must be evaluated using Monte Carlo. Furthermore, because p-KGFN is node-specific, generating candidates for each node requires solving separate p-KGFN optimization problems for each node, further compounding the computational challenges.

4 Our Method

With the challenge of optimizing the p-KGFN acquisition function in mind from the previous section, we introduce in this section a significantly faster-to-optimize acquisition function, Fast p-KGFN, that uses significantly less compute than the original p-KGFN while still providing a similar ability to find good solutions to the original optimization problem using a small number of low-cost partial evaluations.

This Fast p-KGFN consists of two key innovative components: (1) fast candidate generation that reduces a complex search over continuous node inputs to simply evaluating the acquisition function on a small discrete set; and (2) fast acquisition function computation, which enables efficient selection of the enumerated point from (1) with the best p-KGFN value. At each iteration, the method requires solving only one continuous optimization problem, and this continuous problem uses an objective function that is significantly more tractable than the original p-KGFN. Algorithm 1 outlines the complete procedure described in this section.

4.1 Fast Candidate Generation

To generate node-specific candidates as inputs for each node in the network, our Fast p-KGFN algorithm follows these steps. In each iteration, given a DAG representing the function network and initial combined observation set $\mathcal{D}_n$, the proposed Fast p-KGFN first fits the posterior distribution for y_K, induced by the conditional posterior distributions of $f_1, \ldots, f_K$. Next, it optimizes the EIFN acquisition function defined in Eq. (2) from the fitted model to obtain a network candidate $\hat{x}_n$. For example, in Figure 1, a network candidate takes the form $\hat{x}_n = (\hat{x}_{n,1}, \hat{x}_{n,2}, \hat{x}_{n,3})$.

The proposed algorithm then draws a realization $\hat{f}_k$ of each function f_k from its GP posterior and combines these sampled functions according to the DAG structure to construct a realization of all node outputs at the network candidate, $\hat{y}_k(\hat{x}_n)$ using a recursive formula similar to Eq. (1). In Figure 1, this step yields the two intermediate outputs $\hat{y}_1(\hat{x}_n)$ and $\hat{y}_2(\hat{x}_n)$, corresponding to evaluations of the sampled function $\hat{f}_1$ at $\hat{x}_{n,1}$ and $\hat{f}_2$ at $\hat{x}_{n,2}$, respectively.

Finally, for each function node f_k, the algorithm constructs node-specific candidates $\hat{z}_{n,k} = (\hat{y}_{\mathcal{J}(k)}(\hat{x}_n), \hat{x}_{n,\mathcal{I}(k)})$ by concatenating the EIFN-generated candidate components $\hat{x}_{n,\mathcal{I}(k)}$ with the simulated intermediate outcomes from the parent nodes $\hat{y}_{\mathcal{J}(k)}(\hat{x}_n)$.

For example, in Figure 1, node-specific candidates take the form $\hat{z}_{n,1} = \hat{x}_{n,1}$, $\hat{z}_{n,2} = \hat{x}_{n,2}$ and $\hat{z}_{n,3} = (\hat{y}_1(\hat{x}_n), \hat{y}_2(\hat{x}_n), \hat{x}_{n,3})$ to be proposed to evaluate at function nodes f_1, f_2 and f_3, respectively.

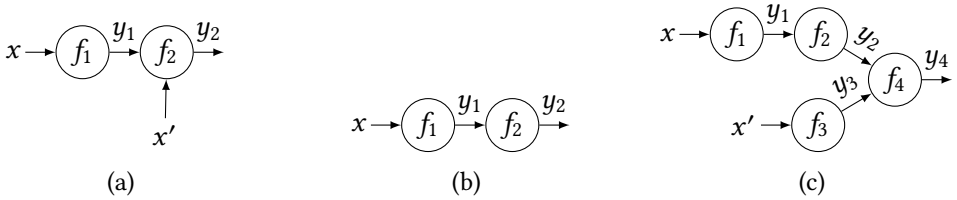

Figure 2: Function networks in the numerical experiments: (a) AckMat (b) FreeSolv and (c) Manu

The p-KGFN acquisition function will then be evaluated for each of these finitely many candidates to identify the one with the largest acquisition function value. To perform this evaluation quickly despite the fact that evaluation requires a nested expectation, we leverage a novel technique described below in Section 4.2. The node $\hat{k}$ with the highest p-KGFN value is then selected for evaluation at the selected input $\hat{z}_{n,\hat{k}}$. The resulting data point $(\hat{z}_{n,\hat{k}}, f_{\hat{k}}(\hat{z}_{n,\hat{k}}))$ is added to the observation set $\mathcal{D}_n$. This process is repeated iteratively until the evaluation budget is depleted.

4.2 Fast Acquisition Function Computation

This section describes an improved approach for computing the p-KGFN acquisition function, which is used as described above in Section 4.1.

Specifically, evaluating the pKGFN acquisition function requires approximating the solutions of the posterior mean optimization problems: $v_n^* = \max_{x\in\mathcal{X}} \nu_n(x)$ and $v_{n+1}^* = \max_{x\in\mathcal{X}} \nu_{n+1}(x)$. To expedite these optimizations, rather than solving them over $\mathcal{X}$, the discretization method solves them over a smaller discrete set $\mathcal{A}$ which is designed to include high-potential solutions. Buathong et al. (2024) proposed two approaches for generating this discrete set $\mathcal{A}$, i.e. (1.) a Thompson sampling-based approach and (2.) local point sampling.

We improve the Thompson sampling-based approach with the following novel strategy. Instead of randomly sampling N_T realizations of the function network from its posterior and optimizing each independently to obtain N_T maximizers for inclusion in $\mathcal{A}$, we draw $M \geq N_T$ realizations from the posterior function network, denoted $\hat{f}_j^{\mathcal{A}}$ (the superscript $\mathcal{A}$ distinguishes this from the realizations $\hat{f}_k$ mentioned earlier in Section 4.1 sampled to construct a network realization $\hat{f}$ used to generate node-specific candidates). Then, we use the sampled realizations together to produce a set S_{N_T} containing N_T potential discrete points as follow:

$$\mathcal{S}_{N_T} \in \underset{\mathcal{S}\subset\mathcal{X}^{N_T}}{\arg\max} \frac{1}{M}\sum_{j=1}^{M} \max_{x\in\mathcal{S}} \hat{f}_j^{\mathcal{A}}(x).$$

This method can be interpreted as a batch Thompson sampling approach, producing a diverse batch of potential solutions for the posterior mean problems. This joint optimization problem offers a faster runtime than optimizing the network realizations individually as in the original p-KGFN.

For local point sampling, we follow the random sampling implementation in Buathong et al. (2024) to generate a set $\mathcal{S}_L$ of N_L local points around the point x_n^*, the maximizer of the current network posterior mean function. A local point $x \in \mathcal{X}$ is defined as one satisfying $d(x, x_n^*) \leq r \max_{i=1,\ldots,d}(b_i - a_i)$, where a_i and b_i are the lower and upper bounds of the input of dimension i^{th} and r is a positive hyperparameter. The discrete set is constructed as $\mathcal{A} = \mathcal{S}_T \cup \mathcal{S}_L$ and is used instead of the original search space $\mathcal{X}$ in solving optimization problems v_n^* and v_{n+1}^*.

4.3 Node Selection

After evaluating the p-KGFN candidates (nodes and input values) from Section 4.1 using discretization approach from Section 4.2, the node $\hat{k}$ with the highest p-KGFN value is then selected for

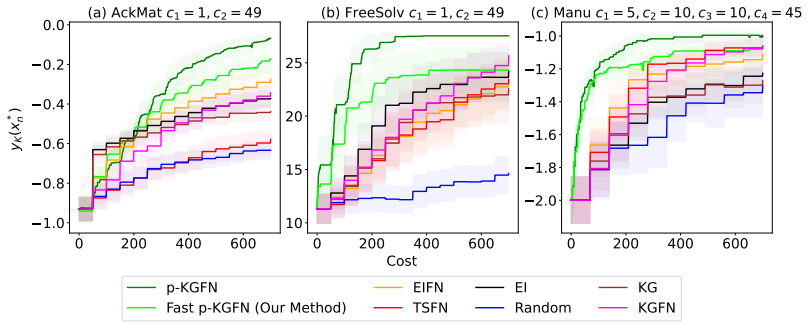

Figure 3: Optimization performance comparing between our proposed Fast p-KGFN algorithm and benchmarks including p-KGFN, EIFN, KGFN, TSFN, EI, KG and Random on three experiments: AckMat (left), Freesolv (middle) Manu (right)

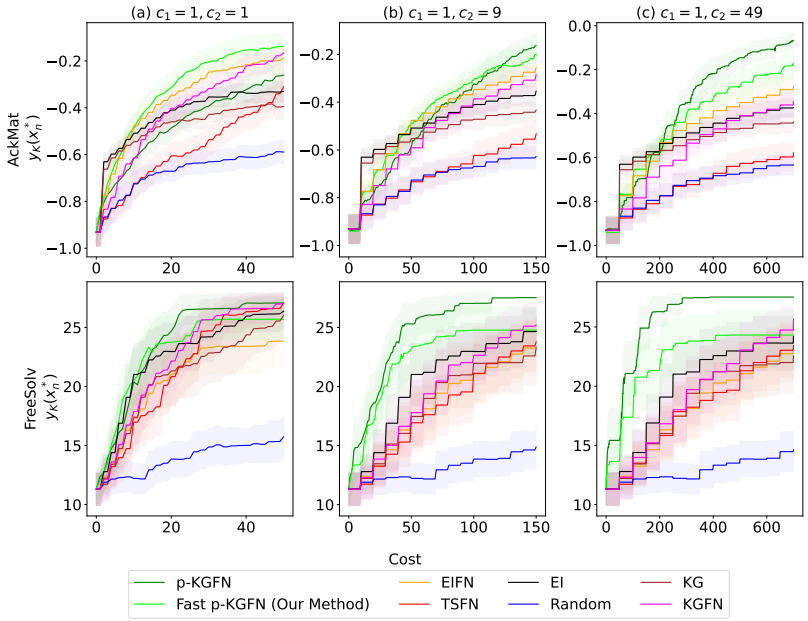

Figure 4: Cost sensitivity analysis for AckMat (Top row) and FreeSolv (Bottom row) problem with different costs (a) $c_1 = 1, c_2 = 1$; (b) $c_1 = 1, c_2 = 9$; and (c) $c_1 = 1, c_2 = 49$.

evaluation at the selected input $\hat{z}_{n,\hat{k}}$. The resulting data point $(\hat{z}_{n,\hat{k}}, f_{\hat{k}}(\hat{z}_{n,\hat{k}}))$ is added to the observation set $\mathcal{D}_n$. This process is repeated iteratively until the evaluation budget is depleted. The complete procedure is outlined in Algorithm 1.

5 Numerical Experiments

This section assesses the efficiency of the proposed algorithm described in Section 4, with parameters $M = N_T = N_L = 10$ and $r = 0.1$ used in the new discretization method. To show that our method provides competitive optimization performance, but offers significantly faster runtime than the original p-KGFN, we consider three test problems previously considered in Buathong et al. (2024): AckMat, FreeSolv and Manu, with structures presented in Figure 2.

AckMat is a synthetic two-node cascade network whose structure is commonly found in real-applications, such as multi-stage simulators and inventory problems. Here, we aim to find optimal values of x and x' which yield the highest value of y_2. FreeSolv (Mobley and Guthrie, 2014) is originally a materials design problem whose objective is to find an optimal small molecule x whose

Table 1: Runtime on 8-core CPUs comparison of the proposed Fast p-KGFN algorithm, p-KGFN, and EIFN on AckMat and FreeSolv problems with three cost scenarios and the Manu problem. The table reports the average runtime (with standard error) over 30 trials and BO courses, along with the speedup factor indicating how many times faster the proposed algorithm is compared to p-KGFN (shown in parentheses)

Runtimes (mins per one BO iteration)				
Problem	p-KGFN	EIFN	Fast p-KGFN (Our method)	(compared to p-KGFN)
(1.a) ActMat $c_1 = 1, c_2 = 1$	11.24 ± 0.45	2.81 ± 0.32	0.98 ± 0.09	(11.47×)
(1.b) ActMat $c_1 = 1, c_2 = 9$	11.75 ± 0.54	0.78 ± 0.08	2.77 ± 0.29	(4.24×)
(1.c) ActMat $c_1 = 1, c_2 = 49$	8.52 ± 0.28	0.69 ± 0.04	1.69 ± 0.15	(5.04×)
(2.a) FreeSolv $c_1 = 1, c_2 = 1$	5.45 ± 0.38	0.53 ± 0.03	0.34 ± 0.02	(16.03×)
(2.b) FreeSolv $c_1 = 1, c_2 = 9$	3.98 ± 0.14	1.70 ± 0.19	0.42 ± 0.02	(9.48×)
(2.c) FreeSolv $c_1 = 1, c_2 = 49$	3.81 ± 0.17	1.00 ± 0.09	0.29 ± 0.02	(13.14×)
(3) Manu $c_1 = 5, c_2 = 10, c_3 = 10, c_4 = 45$	7.80 ± 0.43	0.52 ± 0.07	1.40 ± 0.10	(5.57×)

negative experimental free energy y_2 is maximized with using a computational energy value y_1 as a pre-screen approximation of y_2. The similar network structure can be found in sequential processes. Manu is a problem representing multiple processes happening in the real manufacturing problems. Here, we want to identify the combinations of materials x and x' which yield the highest value of final product y_4.

We consider the cost scenarios: $c_1 = 1$ and $c_2 = 49$ for AckMat and FreeSolv problems, and $c_1 = 5$, $c_2 = 10$, $c_3 = 10$ and $c_4 = 45$ for Manu problem. The BO evaluation budget is set to 700. We defer the full descriptions of these problems to Appendix A.

5.1 Benchmarks and Comparison Metric

We evaluate the proposed algorithm's optimization performance against several baseline methods, including standard Expected Improvement (EI) (Jones et al., 1998; Močkus, 1975), Knowledge Gradient (KG) (Frazier et al., 2008; Wu and Frazier, 2016), and simple random sampling (Random), which do not utilize network structure. Additionally, we compare to EIFN (Astudillo and Frazier, 2021a), Thompson Sampling for Function Networks (TSFN), and Knowledge Gradient for Function Networks (KGFN), which exploit network structure but require full evaluations. We also include the original p-KGFN algorithm, which supports partial evaluations and leverages network structure.

All algorithms start with the same uniformly sampled $2d+1$ initial observations, fully evaluated across the network, where d is the network dimension. Algorithms proceed until the budget of 700 is exhausted. Performance is averaged over 30 trials, each with different initial observations. We report at each iteration the average ground truth value $y_K(x_n^*)$, where $x_n^* \in \arg\max_{x \in \mathcal{X}} \nu_n(x)$, with confidence intervals, as in Buathong et al. (2024). Though EI, KG and Random do not leverage network structure in their sampling strategies, this metric is computed using a model that incorporates it. We also report the CPU runtimes of each algorithm to validate our claim that the proposed algorithm is faster than the original p-KGFN.

All algorithms are implemented in *BoTorch* package (Balandat et al., 2020) in Python. Code to reproduce our numerical experiments can be found at the anonymous repository: https://anonymous.4open.science/r/fastpKGFN-E719/README.md.

6 Results

Figure 3 illustrates the performance of the proposed Fast p-KGFN algorithm compared to various benchmarks on the AckMat (left), FreeSolv (middle), and Manu (right) problems. Among the evaluated methods, all except p-KGFN and the proposed approach require full evaluations. Notably, only EIFN, p-KGFN, and the proposed algorithm incorporate network structure into their sampling strategies. Additionally, both p-KGFN and the proposed method enable partial evaluations, allowing intermediate nodes to be evaluated at any feasible inputs within their known ranges.

Overall, p-KGFN demonstrated the most promising optimization performance, followed closely by the proposed algorithm. This outcome is expected, as p-KGFN fully optimizes the acquisition functions and selects the most promising node-specific candidate at each iteration. While the proposed algorithm also supports partial evaluations, it reuses EIFN candidates, which may slightly reduce candidate quality and lead to a minor performance trade-off. Nevertheless, the proposed method outperforms the original EIFN algorithm, highlighting the benefits of partial evaluations.

To further examine the behavior of the proposed method, we investigate the impact of evaluation costs. Specifically, we vary the evaluation cost of the second node in the AckMat and FreeSolv problems, considering three settings: (a) $c_1 = c_2 = 1$, (b) $c_1 = 1, c_2 = 9$, and (c) $c_1 = 1, c_2 = 49$. These are with the BO evaluation budgets equal to 50, 150 and 700, respectively. Figure 4 presents the effect of evaluation costs on these problems. The results show that the proposed algorithm behaves similarly to p-KGFN, with the benefits of partial evaluations becoming more pronounced as the second node's evaluation cost increases. This allows the p-KGFN and the proposed method to find better solutions more efficiently than other competitors. These findings confirm that despite slightly lower-quality node-specific candidates, the proposed algorithm effectively leverages partial evaluations to achieve strong optimization performance.

Beyond optimization performance, the proposed algorithm offers significant reduction in computational time to the original p-KGFN. Table 1 summaries the average runtimes on 8-core CPU over 30 trials for all experiments and cost scenarios, focusing exclusively on the proposed algorithm, EIFN, and p-KGFN. The proposed method consistently outperformed p-KGFN in runtime, achieving the fastest improvement in FreeSolv with $c_1 = c_2 = 1$, where it attained a 16.03× speedup.

7 Conclusion

In this work, we address the Bayesian optimization of function networks with partial evaluations (p-KGFN), a framework designed for optimizing expensive objective functions structured as function networks, where each node can be queried independently with varying evaluation costs. While p-KGFN has demonstrated significant improvements over traditional approaches by achieving better solutions with lower evaluation budgets, it suffers from high computational overhead due to its reliance on Monte Carlo simulations and the need to solve individual acquisition function problems for node-specific candidate selection in each iteration.

To overcome these challenges, we propose a faster variant of p-KGFN that leverages the expected improvement for function networks (EIFN) framework. Instead of solving separate acquisition problems for each node, the proposed method generates a single candidate for the entire network using EIFN and combines it with simulated intermediate outputs from the surrogate model to generate node-specific candidates. A cost-aware selection strategy then determines which node and corresponding input candidate to evaluate at each iteration.

We evaluate the efficiency of the proposed method across multiple test problems and cost scenarios. The results demonstrate that our approach achieves competitive optimization performance compared to p-KGFN while significantly reducing computational time, with the fastest runtime improvement exceeding 16×.

Despite these advantages, our method has some limitations. It assumes that intermediate nodes can be evaluated at any feasible input without requiring the outputs of parent nodes in advance, which may limit its applicability in more constrained function network scenarios. Extending the method to handle upstream-downstream dependencies is an important direction for future work.

8 Broader Impact Statement

After careful reflection and consideration, the authors believe that this work presents no notable negative impacts to the society or the environment.

References

Ackley, D. (2012). *A connectionist machine for genetic hillclimbing*, volume 28. Springer science & business media.

Astudillo, R. and Frazier, P. (2019). Bayesian optimization of composite functions. In *International Conference on Machine Learning*, pages 354–363. PMLR.

Astudillo, R. and Frazier, P. (2021a). Bayesian optimization of function networks. *Advances in neural information processing systems*, 34:14463–14475.

Astudillo, R. and Frazier, P. I. (2021b). Thinking inside the box: A tutorial on grey-box bayesian optimization. In *2021 Winter Simulation Conference (WSC)*, pages 1–15. IEEE.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538.

Buathong, P., Wan, J., Astudillo, R., Daulton, S., Balandat, M., and Frazier, P. I. (2024). Bayesian optimization of function networks with partial evaluations. In *Proceedings of the 41st International Conference on Machine Learning*, pages 4752–4784.

Cosenza, Z., Astudillo, R., Frazier, P. I., Baar, K., and Block, D. E. (2022). Multi-information source bayesian optimization of culture media for cellular agriculture. *Biotechnology and bioengineering*, 119(9):2447–2458.

Deneault, J. R., Chang, J., Myung, J., Hooper, D., Armstrong, A., Pitt, M., and Maruyama, B. (2021). Toward autonomous additive manufacturing: Bayesian optimization on a 3d printer. *MRS Bulletin*, 46:566–575.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Frazier, P. I., Powell, W. B., and Dayanik, S. (2008). A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439.

Frazier, P. I. and Wang, J. (2016). Bayesian optimization for materials design. *Information science for materials discovery and design*, pages 45–75.

Garnett, G. P. (2002). An introduction to mathematical models in sexually transmitted disease epidemiology. *Sexually transmitted infections*, 78(1):7–12.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276.

Jamil, M. and Yang, X.-S. (2013). A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492.

Kusakawa, S., Takeno, S., Inatsu, Y., Kutsukake, K., Iwazaki, S., Nakano, T., Ujihara, T., Karasuyama, M., and Takeuchi, I. (2022). Bayesian optimization for cascade-type multistage processes. *Neural Computation*, 34(12):2408–2431.

Mobley, D. L. and Guthrie, J. P. (2014). Freesolv: a database of experimental and calculated hydration free energies, with input files. *Journal of computer-aided molecular design*, 28:711–720.

Močkus, J. (1975). On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference: Novosibirsk, July 1–7, 1974*, pages 400–404. Springer.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Sha, D., Ozbay, K., and Ding, Y. (2020). Applying bayesian optimization for calibration of transportation simulation models. *Transportation Research Record*, 2674(10):215–228.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Wu, J. and Frazier, P. (2016). The parallel knowledge gradient method for batch bayesian optimization. *Advances in neural information processing systems*, 29.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We accurately mentioned our contributions in both abstract and introduction

   (b) Did you describe the limitations of your work? [Yes] We mentioned the limitation of our method at the end of Section 7 Conclusion on page 9.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 8

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? (see `https://2022.automl.cc/ethics-accessibility/`) [Yes] We have read the guidelines.

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources, etc.)? [Yes] This is mentioned in Section 5.1 Benchmarks and Comparison Metric

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning details and results, etc.)? [Yes] This is mentioned across Section 5. More details can be found in Appendix A.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] Each problem was repeated for 30 runs with different sets of initial observations as mentioned in Section 5.1

   (d) Did you report the uncertainty of your results (e.g., the standard error across random seeds or splits)? [Yes] BO progress curves and runtimes are reported with standard errors computed from 30 runs. See the figures and the table in Section 6.

   (e) Did you report the statistical significance of your results? [N/A]

   (f) Did you use enough repetitions, datasets, and/or benchmarks to support your claims? [Yes] We considered three different problems with multiple cost scenarios as described in Section 5.

   (g) Did you compare performance over time and describe how you selected the maximum runtime? [Yes] Discussion about runtime can be found in Section 6.

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This can be found in Table 1.

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We did cost sensitivity analysis. See Section 6.

3. With respect to the code used to obtain your results...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all dependencies (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation instructions, and execution commands (either in the supplemental material or as a URL)? [Yes] We provided all the codes required to run the experiments in the anonymous repository mentioned in Section 5.1. We also uploaded the repository as a .zip file on the submission system. One can create an environment with all required python packages using fast_pKGFN_evn.yml file provided in the repository.

README in the repository also includes descriptions of required packages and explanations of how to run the example codes.

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] One can reduce the BO budget for shorter runtime.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] Details can be found in README.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [No] Unfortunately, we cannot provide the raw result data due to its large size, which exceeds the upload limits for GitHub. However, the data can be reproduced by running the code as outlined in the README.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] Although we do not provide the raw data, we offer the source codes used to generate all the figures and tables presented in the paper. These codes are located in the Visualization folder within the anonymous repository. After running the experiments, the "results" folder will be automatically created. One can then load the saved results and reproduce the result figures and tables using the following files: read_results_and_plot_graphs.ipynb (for figures) and read_wallclock.ipynb (for tables).

4. If you used existing assets (e.g., code, data, models)...

(a) Did you cite the creators of used assets? [Yes] All codes that we developed from are cited in the main paper.

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A]

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you created/released new assets (e.g., code, data, models)...

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] See the code

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] We included the anonymous GitHub repository in Section 5.1 and also uploaded it as .zip file on the submission system.

6. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to institutional review board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7. If you included theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [N/A]

(b) Did you include complete proofs of all theoretical results? [N/A]

A Test Problem Descriptions

A.1 Problem 1: ActMat

We consider a two-node cascade network, as illustrated in Figure 2a. The first node is a Ackley function (Ackley, 2012) that takes a 6-dimensional input x as input:

$$f_1(x) = -20\exp\left(-0.2\sqrt{\frac{1}{6}\sum_{i=1}^{6}x_i^2}\right) - \exp\left(\frac{1}{6}\sum_{i=1}^{6}\cos(2\pi x_i)\right) + 20 + \exp(1),$$

Each dimension of x lies in $[-2, 2]$. The second node processes the first node output denoted as y_1 with an additional parameter $x' \in [-10, 10]$ to produce the final output y_2. We use the negated Matyas function (Jamil and Yang, 2013) for this second node:

$$f_2(y_1, x_7) = -0.26(y_1^2 + x_7^2) + 0.48 y_1 x_7,$$

We assume the range of $y_1 \in [0, 20]$. This bound is used by p-KGFN algorithm to generate candidates for the second node.

The evaluation costs for nodes are set as $c_1 = 1$ and $c_2 = 49$. The objective is to find x and x' that maximize the final output y_2.

A.2 Problem 2: FreeSolv

FreeSolv is a molecular design test case built upon the available data set (Mobley and Guthrie, 2014), consisting of calculated and experimental hydration-free energies of 642 small molecules. The problem is constructed as a two-stage function network presented in Figure 2b. Here, the input $x \in [0, 1]^3$ is a continuous representation of small molecule extracted from a variational autoencoder model (Gómez-Bombarelli et al., 2018) and compressed by the principal component analysis. f_1 represents a mathematical model that takes input x and is used to estimate the negative calculated free energy. The second node represents a wet-lab experiment that takes this calculated energy output as input and returns the negative experimental free energy, a target we aim to maximize. To construct this continuous network optimization problem, two GP models for calculated and experimental energies are separately fitted using all available data and the posterior mean functions of these two GP models are used as the functions f_1 and f_2, respectively.

We estimate the range of y_1 from the raw dataset and assume that $y_1 \in [-5, 30]$. Similar to AckMat problem, the evaluation costs are set to be $c_1 = 1$ and $c_2 = 49$.

A.3 Problem 3: Manufacturing (Manu)

A manufacturing problem is constructed as the function network shown in Figure 2c. Each function is drawn from a GP prior with Matérn 5/2 kernels and varying length scale parameters to reflect the different complexities of the individual process. The length scale parameters are 0.631, 1, 1 and 3 for f_1, f_2, f_3 and f_4, respectively. The outputscale parameter is set to 0.631 for all functions, except f_4 which uses 10.

We assume the intermediate outputs' ranges as follows: $y_1 \in (-2, 2)$ and $y_2, y_3 \in (-1, 1)$,. The inputs x and x' are constrained to $(-1, 1)$. Evaluation costs are assigned as $c_1 = 5$, $c_2 = 10$, $c_3 = 10$ and $c_4 = 45$. The goal is to determine the optimal pair of raw materials x and x' that maximize the final output y_4.